CHALMERS

Utveckling av en Web API-baserad Läroplattform för TestScouts University

Degree Project Report in Computer Engineering

Sourena Samimiamoli
Yosuf Amiri

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Degree project report 2025

Utveckling av en Web API-baserad Läroplattform för TestScouts University

Sourena Samimiamoli
Yosuf Amiri

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2025

Utveckling av en Web API-baserad Läroplattform för TestScouts University
Development of a Web API-based Learning Platform for TestScouts University

Sourena Samimiamoli
Yosuf Amiri

© Sourena Samimiamoli, 2025.
© Yosuf Amiri, 2025.

Supervisor: Henrik Jansson Valter, Department of Computer Science and Engineering
Examiner: Nicholas Smallbone, Department of Computer Science and Engineering

Degree project report 2025
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Utveckling av en Web API-baserad Läroplattform för TestScouts University
Development of a Web API-based Learning Platform for TestScouts University

Sourena Samimiamoli
Yosuf Amiri
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

This thesis aims to develop a foundational version of *TestScouts University*, a web-based platform that provides educational resources and guidance for QA professionals. The platform is built on a .NET-based Web API with MongoDB for data storage, and is complemented by a React-based frontend. The system includes user management, course progression, road-map structures, and a CI/CD pipeline via GitLab CI. The goal is to deliver a minimal yet functional system that can be extended further beyond the scope of this thesis.

Utveckling av en Web API-baserad Läroplattform för TestScouts University

Examensarbete
Författare: Sourena Samimiamoli
Yosuf Amiri
Handledare: Henrik Jansson Valter

Maj 2025

Abstract

This thesis aims to develop a foundational version of *TestScouts University*, a web-based platform that provides educational resources and guidance for QA professionals. The platform is built on a .NET-based Web API with MongoDB for data storage, and is complemented by a React-based frontend. The system includes user management, course progression, roadmap structures, and a CI/CD pipeline via GitLab CI. The goal is to deliver a minimal yet functional system that can be extended further beyond the scope of this thesis.

Innehåll

1 Inledning **3**
1.1 Bakgrund . . . 3
1.2 Syfte och mål . . . 3
1.3 Avgränsningar . . . 3

2 Teknisk Bakgrund **4**
2.1 Backend och C# .NET . . . 4
2.2 Databas: MongoDB . . . 5
2.3 Autentisering med JWT och OAuth . . . 5
2.4 CI/CD med GitLab . . . 5
2.5 React Frontend . . . 6
2.5.1 React Vite som utvecklingsverktyg . . . 7
2.5.2 React Router . . . 8
2.5.3 UI-ramverk: CSS och React Bootstrap . . . 8
2.5.4 Ikonbibliotek . . . 8
2.5.5 React Circular Progressbar . . . 8

3 Metod **9**
3.1 Utvecklingsstrategi och val av teknisk stack . . . 9
3.2 Planering av datamodell och arkitektur . . . 9
3.3 Val av autentiseringsstrategi . . . 10
3.4 Teststrategi och kvalitetssäkring . . . 11

4 Systemkonstruktion **11**
4.1 Systemstruktur . . . 11
4.2 Databasdesign och datastruktur . . . 12
4.3 Frontendstruktur och användargränssnitt . . . 12

4.4 Medieintegration och materialvisning . . . 12
4.5 Autentisering och säkerhetslösning . . . 12
4.6 Testning och kvalitetssäkring . . . 13
4.7 Klassdiagram . . . 13
4.8 Sekvensdiagram . . . 13
4.9 Sammanfattning . . . 15

5 Testning och Kvalitetssäkring **15**
5.1 Enhetstestning . . . 16
5.2 CI/CD och testautomatisering . . . 17
5.3 Manuell testning av frontend . . . 17

6 Resultat **17**

7 Diskussion **18**

8 Framtida arbete **19**

1 Inledning

1.1 Bakgrund

Inom mjukvarutestning finns ett växande behov av strukturerade inlärningsresurser för att stödja utbildning av nya och erfarna QA-ingenjörer. Företaget TestScouts AB såg behovet av en intern plattform – TestScouts University"– som skulle kunna samla kurser, vägledningar och annan utbildningsdokumentation inom ett centralt system.

1.2 Syfte och mål

Syftet med detta examensarbete är att utveckla en grundläggande, funktionell och skalbar utbildningsplattform i form av ett webbaserat system. Plattformen riktar sig till användare inom företaget TestScouts och ska fungera som ett stöd för lärande, övning och vägledning inom testning och kvalitetssäkring. Projektets fokus ligger på att skapa ett stabilt backend-API, integrera detta med en modern frontend samt etablera automatiserad testning och CI/CD-flöde för kontinuerlig leverans.
Målet är att leverera en första version av systemet (MVP – Minimum Viable Product), som innehåller stöd för:

- Hantering av användare, kurser och utbildningsvägar via ett REST-baserat backend-API utvecklat i C#/.NET.
- En frontend utvecklad i React som kommunicerar med backend och presenterar kursinnehåll och progression.
- Autentisering med stöd för både lokala inloggningar och Google OAuth.
- En CI/CD-pipeline i GitLab med automatiska tester och validering av kod vid varje ändring.
- Dokumentation och struktur som möjliggör framtida vidareutveckling och utbyggnad.

Genom att ta fram denna helhetslösning skapas ett tekniskt fundament som kan vidareutvecklas för att stödja mer avancerade funktioner i framtiden.

1.3 Avgränsningar

För att hålla projektet fokuserat och genomförbart inom ramen för examensarbetets omfattning har följande avgränsningar tillämpats:

- Projektet omfattar endast en första version (MVP) med grundläggande funktionalitet – t.ex. inloggning, kurshantering och progression.
- Mer avancerade funktioner såsom statistik, rekommendationssystem, admin-gränssnitt och rollstyrd åtkomst implementeras ej inom detta arbete, men förbereds strukturellt för framtida utbyggnad.
- Endast teknologier som .NET, MongoDB, React och GitLab CI har använts, för att säkerställa stabilitet och minska komplexitet.
- All data som används i systemet är testdata och inte kopplat till faktiska utbildningsprogram inom företaget.

Dessa avgränsningar möjliggör att projektets mål uppnås inom given tidsram, utan att kompromissa med kvalitet eller teknisk grund.

2 Teknisk Bakgrund

Detta projekt utgör backend-delen av en webbaserad läroplattform kallad TestScouts University. Målet med backend är att tillhandahålla ett robust REST API byggt i ASP.NET Core 8, som hanterar användare, kurser, utbildningsvägar (roadmaps), och kursprogression. API:et kommunicerar med en MongoDB-databas för lagring och hämtning av data och är säkrat med JWT-baserad autentisering, med stöd för både lokal inloggning och Google OAuth. Plattformen är byggd enligt moderna arkitekturprinciper som separation av ansvar, beroendeinjektion, och skalbarhet via mikroservicevänlig struktur. Fokus ligger på ett funktionellt, säkert och utbyggbart backend-system.

2.1 Backend och C# .NET

Backendsystemet för *TestScouts University* är utvecklat i C# med ramverket ASP.NET Core 8. Valet av .NET som plattform grundar sig i dess stabilitet, höga prestanda och breda stöd för att bygga skalbara webbtjänster [1]. Systemet tillhandahåller ett webb-API som möjliggör kommunikation mellan frontend och backend, där all datalagring, autentisering och logik hanteras server-side. Den utvecklade backenddelen utgör grunden för ett utbildningssystem som riktar sig mot testingenjörer och QA-personal inom företaget TestScouts.

API-design

Webb-API:et använder en REST-arkitektur och är uppbyggt kring tydligt definierade endpoints som möjliggör klassiska CRUD-operationer – *Create, Read, Update* och *Delete* – på centrala resurser som kurser, kapitel, användare och roadmap-strukturer. API:et är utformat för att kommunicera asynkront med frontendapplikationen genom JSON-baserad datatrafik, vilket ger snabb och effektiv datahantering i användargränssnittet. Exempel på vanliga operationer är att hämta tillgängliga kurser, uppdatera användarprogression samt autentisera användare via e-post eller Google-inloggning[2].

Databehandling och validering

Backendlogiken hanterar och skyddar data i enlighet med goda säkerhetsprinciper. Exempelvis krypteras alla användarlösenord innan de lagras i databasen med hjälp av `BCrypt`, vilket säkerställer att inga känsliga uppgifter sparas i klartext. API:et är också byggt för att skydda känsliga datafält från att skickas till klienten – autentiseringstokens, lösenord och interna ID-maskeras eller utesluts helt i API-responsen. Systemet har inbyggd åtkomstkontroll där endpoints kräver giltiga JWT-tokens, vilket innebär att endast autentiserade användare kan interagera med skyddade resurser. Datavalidering sker både på klient- och serversidan, och API:et tillämpar en strukturerad hantering av fel och undantag för att ge tydlig återkoppling vid felaktiga anrop.

Databasinteraktion

Backendsystemet är direkt kopplat till en MongoDB-databas där all strukturell data lagras i dokumentform. Genom att använda MongoDB kan systemet hantera kurser, roadmap-strukturer, användarprogression och annan relaterad information utan behov av traditionella relationsscheman. Interaktionen med databasen sker via MongoDB:s officiella .NET-drivrutin, och API:et utför olika operationer för att hämta, filtrera och uppdatera data. Exempelvis kan systemet returnera alla kurser inom ett roadmap-spår eller markera vilka kapitel en användare har slutfört [3]. Datamodellerna i backend speglar strukturen i databasen, vilket underlättar insamling, bearbetning och överföring av information mellan lagren i applikationen.

2.2 Databas: MongoDB

MongoDB används som den primära databasen i projektet. Denna NoSQL-databas lagrar information i JSON-liknande dokument, vilket ger hög flexibilitet och möjlighet att snabbt iterera på datastrukturerna under utvecklingsfasen. Till skillnad från relationsdatabaser som kräver fördefinierade tabeller och kolumner, tillåter MongoDB dynamiska datamodeller, vilket passar utmärkt för applikationer där strukturen på objekten kan förändras över tid. I *TestScouts University* används MongoDB för att lagra information om användare, kurser, kapitel, samt roadmap-konfigurationer. Med hjälp av index och textbaserad sökning kan användaren snabbt filtrera och hitta rätt innehåll, även med delvis matchande sökord.

2.3 Autentisering med JWT och OAuth

Autentiseringen i systemet bygger på JSON Web Tokens (JWT), vilket innebär att varje användare som loggar in tilldelas en signerad token som fungerar som en nyckel för att identifiera och verifiera användaren i efterföljande API-anrop. Detta möjliggör så kallad *stateless*-autentisering där servern inte behöver hålla aktiva sessioner, vilket är både skalbart och säkert [4]. Projektet har stöd för två autentiseringsflöden: klassisk inloggning med e-post och lösenord samt OAuth-baserad inloggning via Google. Det senare ger användaren möjlighet att logga in med sitt Google-konto utan att behöva skapa ett nytt konto i plattformen. JWT-token genereras efter lyckad inloggning och innehåller information om användarens identitet, roll och giltighetstid. Säkerhetspolicyn för autentisering hanteras genom middleware i .NET, vilket innebär att API:et automatiskt verifierar att token är giltig vid varje skyddat anrop.

2.4 CI/CD med GitLab

CI/CD (Continuous Integration/Continuous Delivery) avser en praxis för att automatisera bygg-, test- och leveransprocesser i mjukvaruutveckling. CI innebär att utvecklare ofta integrerar (mergar) kodförändringar i en gemensam huvudgren med automatiserade byggen och tester som körs vid varje incheckning[5]. CD (continuous delivery/deployment) utökar detta så att den testade koden kan levereras till produktion på ett säkert och upprepningsbart sätt med minimal manuell hantering[6]. Genom att automatisera tester och distribution möjliggör CI/CD snabbare iterationer och tidig upptäckt av fel, vilket ökar kodkvalitet och minskar riskerna vid utrullning[5].

GitLab definieras CI/CD-pipelines med en YAML-fil `.gitlab-ci.yml` i projektets repository. En pipeline består av steg (stages) och jobb (jobs) som körs av GitLab-runners. Varje jobb specificerar vilka kommandon som ska utföras (t.ex. bygg eller test) och kan köras i valfri Docker-image. GitLabs dokumentation förklarar att pipelines körs för definierade händelser (t.ex. push eller merge request) och att jobb är ordnade i sekvens via stages[7]. Om alla jobb i ett stage lyckas går pipelinen vidare till nästa stage[7]. I vårt projekt har man skilt på frontend och backend: Till exempel används en Node-basbild för frontend där kommandon som `npm run test` körs för att exekvera enhetstester. För backend `C#/.NET` körs sannolikt `dotnet build` och `dotnet test`, där testresultaten exporteras som XUnit-XML-filer. GitLab kan sedan läsa dessa via reports: Xunit i YAML-konfigurationen[8], vilket gör att testresultaten presenteras direkt i GitLab-gränssnittet.

Kanarieprincipen innebär en gradvis och begränsad utrullning av ny kod. Istället för att direkt ersätta hela systemet med en ny version uppdateras först endast en liten del av tjänsten eller användarbasen (kanariedelen), medan övriga använder den tidigare versionen. Om den nya versionen innehåller fel påverkas då endast denna begränsade skara, vilket gör det möjligt att snabbt fixa eller återställa uppdateringen[9]. På så sätt fungerar kanarieutgåvor som en sorts tidig varning – om allt går bra kan man successivt släppa ut nya förändringar till hela systemet[10]. I vårt projekt är det inte specificerat hur kanarieprincipen tillämpats, men en möjlig användning hade kunnat vara att först distribuera uppdateringar till en test- eller staging-miljö

och observera systemets beteende innan full produktion. Även om GitLab erbjuder stöd för avancerad trafikstyrning ("Canary Ingress" i Kubernetes)[9]. sker troligen här en enklare tillämpning genom separata miljöer eller manuell granskning för att gradvis minska risken vid driftsättning.

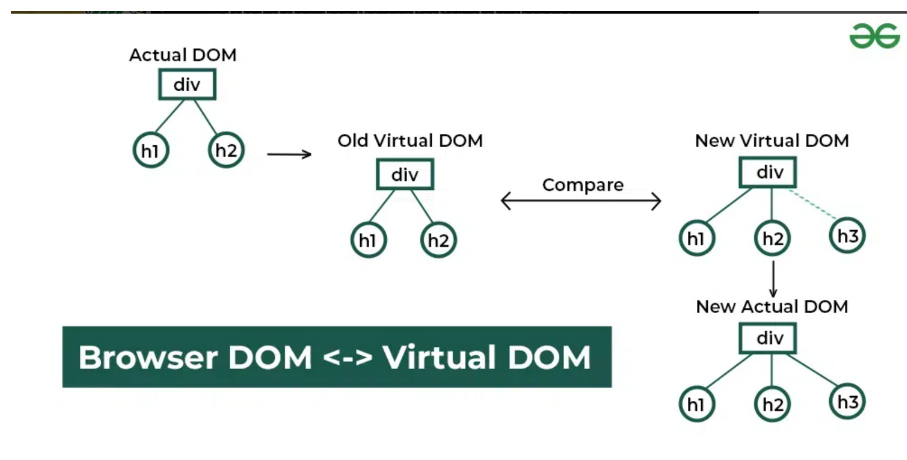
För att automatisera testning och distribution av backendtjänsten används en CI/CD-pipeline konfigurerad via GitLab. I projektets rotkatalog finns en `.gitlab-ci.yml`-fil som definierar hur koden ska byggas, testas och eventuellt distribueras vid varje ändring i versionshanteringssystemet. CI/CD-processen är uppdelad i flera faser: först verifieras att projektet kompilerar korrekt, därefter körs eventuella tester, och slutligen skapas *artifacts* som kan användas för vidare steg i pipelineflödet eller distribution.

Denna automatisering säkerställer att koden hålls i ett fungerande tillstånd och att alla ändringar som skickas till GitLab-grenen genomgår grundläggande kvalitetssäkring.

2.5 React Frontend

React är ett JavaScript-bibliotek för att bygga användargränssnitt. Det utvecklades av Meta (tidigare Facebook) och introducerades 2013 **react**. React möjliggör komponentbaserad utveckling, vilket innebär att gränssnittet konstrueras av oberoende och återanvändbara byggblock **react_docs**. Denna struktur gör det enklare att organisera kod och underlätta vidareutveckling [11].

En central del av Reacts funktion är användningen av en virtuell DOM (Document Object Model), som är en representation av gränssnittet i minnet **react_virtualdom**. Istället för att uppdatera hela sidan vid varje förändring, identifierar React vilka delar som faktiskt har ändrats och uppdaterar endast dessa **react_diff**. Detta kan leda till förbättrad prestanda, särskilt i applikationer med mycket interaktivitet [12].



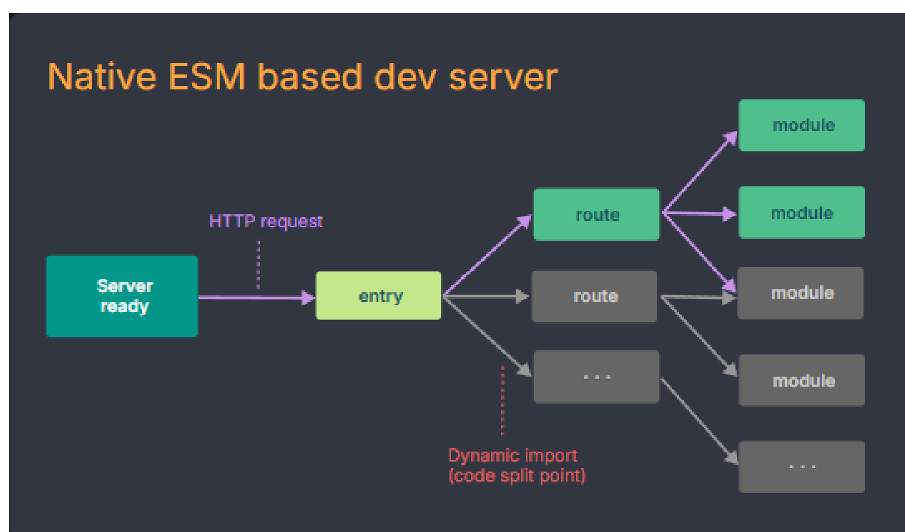
Figur 1: Reacts virtuella DOM uppdaterar endast förändrade noder vid rendering

Flera tekniska artiklar och översikter lyfter fram att den komponentbaserade arkitekturen bidrar till tydligare struktur, ökad återanvändbarhet och enklare samarbete vid parallell utveckling **react_modularity**. Reacts diff-algoritm hjälper till att hålla gränssnittet konsekvent genom att selektivt uppdatera DOM-noder snarare än att rita om hela sidan **react_diff**.
React använder JSX (JavaScript XML), en syntaxförlängning som låter utvecklare skriva HTML-liknande kod direkt i JavaScript. Detta kan förenkla utvecklingen genom att kombinerar markup och logik i samma komponent **react_docs**. JSX är deklarativt, vilket innebär att utvecklaren beskriver önskat gränssnittstillstånd snarare än exakta steg för att nå det.
React är idag ett vanligt förekommande verktyg inom webbapplikationsutveckling. Enligt Stack

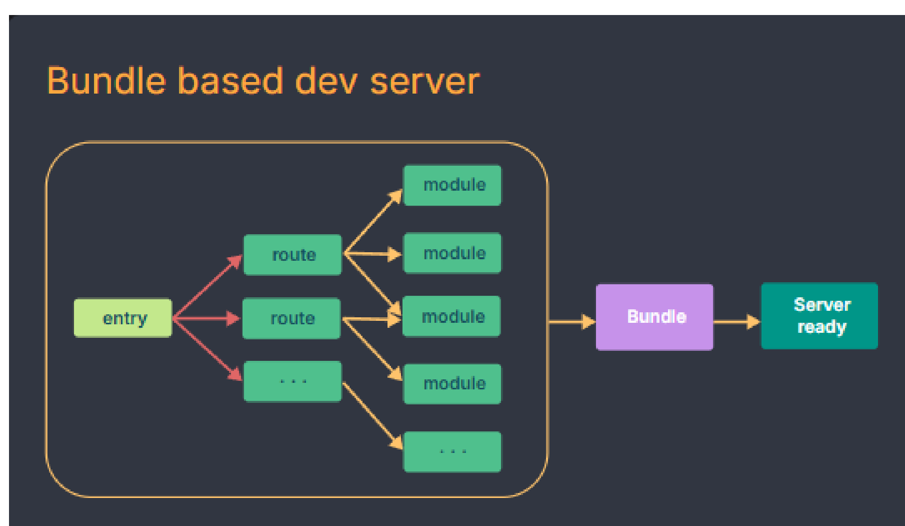
Overflow:s årliga utvecklarundersökning 2023 rapporterade 41,6 % av de professionella utvecklarna att de arbetat med React under det senaste året **stackoverflow2023**, vilket visar att det fortsatt är en relevant teknik i branschen.

2.5.1 React Vite som utvecklingsverktyg

Vite är ett bygg- och utvecklingsverktyg för moderna webbprojekt, lanserat 2020 av skaparen av Vue.js **vite**. Verktyget är framtaget för att optimera utvecklingsflödet med snabb projektstart, modulär hantering av kod och korta byggtider. Till skillnad från traditionella bundlare som Webpack paketerar Vite inte om hela applikationen vid varje förändring. Istället används webbläsarens inbyggda stöd för ES-moduler och Vite serverar endast den kod som ändrats **vite_docs**. Vite använder bland annat *esbuild*, ett verktyg skrivet i Go, för att snabbt förkompilera beroenden. Detta gör att projektstart kan gå 10–100 gånger snabbare jämfört med äldre verktyg [13]. Applikationens egen kod levereras som fristående moduler till webbläsaren, vilket innebär att resultatet av ändringar kan ses omedelbart när utvecklingsservern körs **vite_hmr**.



Figur 2: Utvecklingsflöde med Vite. Endast nödvändig kod laddas vid ändringar.



Figur 3: Traditionellt flöde med Webpack, där hela projektet paketeras om vid varje ändring.

Utvecklarupplevelsen påverkas även av verktygets konfiguration. Vite använder principen

”zero-config”, vilket innebär att vanliga fall konfigureras automatiskt. Detta minskar behovet av manuell inställning **vite_docs**. Exempelvis visar DigitalOcean att ett nytt React-projekt med Create React App ofta har runt 140 MB beroenden, medan motsvarande Vite-projekt bara kräver cirka 31 MB **digitalocean_vite**. Enligt vissa analyser kan detta leda till snabbare installation, enklare uppstart och smidigare arbetsflöde.

I detta projekt används Vite via kommandot `npm run dev`. Tack vare Vites upplägg kan utveckling ske med snabb återladdning av gränssnittet utan att hela applikationen kompileras om. Detta ger kortare väntetider och förenklar testning av gränssnittsförändringar.

2.5.2 React Router

React Router är ett bibliotek för att hantera navigering i React-applikationer. Det möjliggör deklarativ routing, vilket innebär att olika URL-vägar kopplas till specifika React-komponenter. Detta gör det möjligt att skapa så kallade single-page-applikationer (SPA), där användaren kan växla mellan olika vyer utan att sidan laddas om [14].

I projektet används React Router för att varje kurs och kapitel ska kunna nås direkt via en unik URL. Detta förenklar navigationen i applikationen och gör det lättare att skapa en överskådlig struktur.

2.5.3 UI-ramverk: CSS och React Bootstrap

Gränssnittet i applikationen är uppbyggt med hjälp av både CSS och komponentbiblioteket React Bootstrap. CSS används för att definiera stilar som färger, typsnitt, marginaler och layout, samt för att göra gränssnittet responsivt över olika skärmstorlekar **css_w3c**. Genom att separera presentation från funktionalitet blir koden mer strukturerad och underhållbar.

React Bootstrap bygger på det etablerade Bootstrap-ramverket, men erbjuder dess komponenter i form av React-komponenter **react_bootstrap**. Det innebär att man kan använda färdiga gränssnittselement som knappar, formulär och menyer direkt i React-koden, utan att behöva implementera interaktivitet med t.ex. jQuery [15].

I detta projekt används dessa verktyg för att skapa ett tydligt och användarvänligt gränssnitt. CSS används där justeringar krävs utöver Bootstraps grundläggande stilklasser, medan React Bootstrap används för komponenter som formulär, knappar och navigation.

2.5.4 Ikonbibliotek

För att hantera ikoner i applikationen används två bibliotek: React Bootstrap Icons och React Icons. React Bootstrap Icons tillhandahåller över 2000 vektorbaserade ikoner från Bootstraps officiella ikonpaket, anpassade som React-komponenter. React Icons är ett meta-bibliotek som samlar ikoner från flera källor, såsom Font Awesome och Material Icons **react_icons**.

Båda biblioteken gör det möjligt att inkludera specifika ikoner med ES6-importer. Detta innebär att endast de ikoner som används inkluderas i projektet, vilket kan minska paketstorlek och bidra till tydligare kod [16].

2.5.5 React Circular Progressbar

React Circular Progressbar är ett komponentbibliotek för att visa cirkulära framstegsindikatorer i React-applikationer. Komponenten är byggd med SVG och kan anpassas med CSS för att visualisera t.ex. procentandelar eller färdigställandegrad [17].

I detta projekt används komponenten för att indikera hur mycket av en kurs eller roadmap användaren har genomfört. Detta ger en visuell återkoppling som kan underlätta översikt och motivation för slutanvändaren.

3 Metod

Detta kapitel beskriver den metodik och de strategiska val som låg till grund för utvecklingen av plattformen. Målet med examensarbetet var att bygga en webbaserad kursplattform med stöd för kursvisning, kapitelstruktur, användarhantering och autentisering. Utöver detta skulle systemet vara lätt att underhålla, testbart, responsivt och byggt med moderna webbutvecklingsramverk. För att uppnå detta tillämpades en iterativ, funktionsdriven metod som kombinerade teknisk forskning, kravanalys, agil arbetsprocess och modulär implementation. Metoden fokuserade på kontinuerlig feedback och successiv förbättring, där både tekniska val och arkitektur anpassades längs vägen baserat på utvärdering och testning.

3.1 Utvecklingsstrategi och val av teknisk stack

I ett inledande skede genomfördes en kartläggning av systemets övergripande krav: användare skulle kunna bläddra bland kurser, läsa kapitel, logga in, registrera sig, samt följa sin progression genom en kurs. Det identifierades tidigt att applikationen skulle ha både en frontend (klientdel) och en backend (serverdel), vilket ledde till beslutet att arbeta med en tydligt uppdelad fullstack-arkitektur. Ett centralt mål med detta var att möjliggöra oberoende utveckling, testning och felsökning av respektive del, samt att lägga grunden för potentiell framtida vidareutveckling.

För backend valdes ASP.NET Core i kombination med C# som primär teknologi. Detta ramverk är väl anpassat för att bygga RESTful API:er, har inbyggt stöd för beroendeinjektion och erbjuder hög prestanda samt god dokumentation [18]. En annan fördel med ASP.NET Core är dess kompatibilitet med moderna säkerhetsstandarder, vilket var centralt för hantering av användardata.

Frontend utvecklades med React, ett komponentbaserat JavaScript-bibliotek som möjliggör återanvändning av UI-komponenter och effektiv rendering. React valdes eftersom det är ett av de mest använda biblioteken inom modern webbutveckling, har stort community-stöd och ger utvecklaren kontroll över applikationens tillstånd och struktur.

Applikationen skulle byggas som en Single Page Application (SPA), vilket innebär att hela användargränssnittet laddas en gång, varefter navigering och dataflöde sker asynkront på klientsidan via API-anrop. Denna strategi valdes för att uppnå snabbare laddningstider, förbättrad användarupplevelse och mer dynamisk presentation av innehåll. Som byggverktyg för frontend användes `Vite`, vilket erbjuder snabb utvecklingsserver, optimerade och snabb omstart, vilket var fördelaktigt under utvecklingscyklerna [19].

För versionshantering användes Git och GitLab, där arbetet organiserades i grenar och merge requests", vilket möjliggjorde en strukturerad och spårbar utvecklingsprocess. CI/CD användes också i GitLab för att automatiskt köra tester vid varje incheckning av kod.

3.2 Planering av datamodell och arkitektur

Ett viktigt metodologiskt steg var planeringen av datamodellen och hur informationen i applikationen skulle struktureras. Vi började med att analysera kärnobjekten i systemet, och kom

snabbt fram till att kursen utgjorde den centrala entiteten, med kapitel som underordnade komponenter. I samråd med litteratur kring datamodellering i dokumentdatabaser, och baserat på systemets funktionella krav, valde vi att strukturera en kurs som ett dokument med en inbäddad lista av kapitel.

Denna modell passar särskilt väl i MongoDB eftersom det möjliggör att relaterad information (t.ex. en kurs och dess kapitel) lagras tillsammans. Detta är en grundläggande princip i dokumentdatabaser: om relaterad data ofta hämtas tillsammans bör den också sparas tillsammans [20]. Vi kunde därmed hämta hela kursstrukturen med ett enda anrop, vilket minimerar behovet av flera round-trips till databasen.

Vi valde också att hålla datamodellen flexibel. MongoDB:s schemalösa egenskaper innebar att vi kunde iterativt lägga till nya fält och anpassa strukturen utan omfattande migreringar. Exempelvis lade vi senare till ett `Tags`-fält för att möjliggöra förbättrad sökbarhet i kurslistan.

Applikationens backend planerades enligt en flerskiktad arkitektur med separation mellan controllers (ansvarar för HTTP-anrop), services (affärslogik) och repositories (dataåtkomst). Vi definierade också tydliga interfaces för våra repository-klasser, vilket möjliggjorde att dessa kunde mockas under testning.

På frontend-sidan planerade vi komponentstrukturen så att varje vy i applikationen (t.ex. kursöversikt, kursdetalj, inloggningssida) skulle ha sina egna underkomponenter. Vi ritade enkla wireframes för att planera hur användarflödena skulle se ut, samt definierade hur data skulle flöda mellan komponenter och API.

3.3 Val av autentiseringsstrategi

Redan i planeringsfasen beslutades att applikationen skulle stödja två autentiseringsmetoder: traditionell inloggning med e-post och lösenord, samt inloggning via Google med hjälp av OAuth 2.0. Detta beslut grundade sig dels i att olika användare föredrar olika inloggningsmetoder, och dels i att OAuth erbjuder ökad säkerhet och användarvänlighet [21].

För lösenordshantering valde vi att använda hashning med saltning. Detta innebär att varje lösenord, innan det lagras i databasen, omvandlas till en hash-sträng tillsammans med en slumpmässig salt-komponent. Hashningen sker med hjälp av en modern algoritm såsom Argon2 eller Bcrypt, båda välkända för att vara resistenta mot attacker som brute force och rainbow tables. Genom detta tillvägagångssätt undviks att känslig information sparas i klartext [22].

OAuth 2.0 implementerades för att möjliggöra Google-inloggning. Denna metod innebär att autentiseringen hanteras av Google, vilket innebär att vi som utvecklare inte behöver hantera användarens lösenord alls. Istället returneras en ID-token som vi verifierar på serversidan. Om tokenen är giltig kan vi identifiera användaren och skapa ett konto kopplat till deras Google-profil.

Sessionshanteringen planerades att bygga på JSON Web Tokens (JWT), där en token genereras efter inloggning och skickas till klienten, där den lagras i exempelvis `localStorage`. Vid varje efterföljande API-anrop inkluderas tokenen i HTTP-headern, vilket möjliggör stateless autentisering. Detta är en etablerad metod som lämpar sig särskilt väl för SPA-arkitekturer, där det inte är praktiskt att spara sessionsinformation på serversidan.

3.4 Teststrategi och kvalitetssäkring

Testning betraktades som en integrerad del av utvecklingsmetodiken och inte som ett fristående steg. Redan vid planeringen definierade vi vilka delar av applikationen som var kritiska att testa – exempelvis affärslogiken kring kursadministration, autentiseringsflödet, samt interaktionen med databasen. Vi bestämde oss för att kombinera enhetstester med integrationstester för att täcka både isolerade funktioner och helhetsflöden.

På backend-sidan planerades enhetstester med hjälp av testbiblioteket `xUnit`, samt användning av `Moq` för att mocka beroenden. Detta innebar att vi kunde testa t.ex. `CourseService` utan att behöva en riktig databas. Genom att definiera interfaces för våra repositories kunde vi injicera falska implementationer och därmed fokusera testningen på affärslogik snarare än externa beroenden [23].

För integrationstestning valde vi att använda ASP.NET Core:s inbyggda stöd för att köra testservere lokalt i minnet. Detta möjliggjorde att vi kunde genomföra fullständiga flöden såsom registrering → inloggning → hämta skyddad resurs, vilket gav högre tillförlitlighet att alla lager fungerade korrekt tillsammans [24].

Frontend-testningen fokuserades initialt på manuell testning under utveckling, i kombination med kodvalidering via webbläsarens utvecklingsverktyg. På sikt planerade vi att implementera automatiska testfall för komponenter och UI-flöden med hjälp av bibliotek som `Jest` eller `React Testing Library`, men detta prioriterades lägre under prototypfasen.

All testning integrerades i vår CI/CD-pipeline, vilket innebar att varje kodändring automatiskt utlöste en testkörning. Detta hjälpte till att upptäcka fel tidigt och säkerställa att ingen ny kod introducerade regressionsproblem i befintlig funktionalitet [25].

4 Systemkonstruktion

4.1 Systemstruktur

Systemet är uppbyggt som en fullstack-webbapplikation där backend och frontend är tydligt separerade men kommunicerar via ett RESTful API. Backend är utvecklad i ASP.NET Core med C# och följer en lagerindelad arkitektur som påminner om MVC-modellen. Detta innebär att olika delar av systemet har separata ansvar, vilket ökar modularitet, testbarhet och underlättar vidareutveckling.

Controllernivån fungerar som gränssnitt mellan klienten (frontend) och backend. HTTP-anrop tas emot av exempelvis `CoursesController`, som vidarebefordrar begäran till motsvarande tjänst, som `CourseService`. Tjänstelagret hanterar affärslogiken och kommunicerar med databasen via så kallade repository-klasser. Dessa är ansvariga för CRUD-operationer mot MongoDB. Modellerna (`CourseModel`, `UserModel` etc.) definierar datastrukturen och fungerar som kontrakt mellan backend och databasen.

Backend är uppbyggd med beroendeinjektion, vilket gör att komponenter enkelt kan bytas ut, mockas för testning och injiceras vid behov. Detta förbättrar systemets underhållbarhet och testbarhet betydligt.

4.2 Databasdesign och datastruktur

För datalagring används MongoDB, en dokumentorienterad NoSQL-databas. Varje kurs representeras som ett dokument som innehåller fält som kursens ID, namn, beskrivning samt en inbäddad lista av kapitel. Varje kapitel innehåller i sin tur metadata som titel, typ av material (t.ex. video eller PDF) och referens till innehållet.

Denna struktur möjliggör att en kurs med alla dess kapitel kan hämtas med ett enda databasanrop, vilket både förenklar API-designen och förbättrar prestandan. Eftersom kapitel alltid hör till en specifik kurs var inbäddning ett naturligt val framför användning av externa referenser.

Databasoperationerna hanteras via MongoDB:s C#-drivrutin och LINQ, vilket gör frågorna mer läsbara. En sökfunktion för kurser implementerades med hjälp av `$regex`-operatorn för att tillåta substring-matchning på fält som namn eller etiketter. I utvecklingsmiljön användes en lokal MongoDB-instans, medan produktionen körs via MongoDB Atlas.

4.3 Frontendstruktur och användargränssnitt

Frontend är utvecklad med React och strukturerad som en Single Page Application (SPA). Det innebär att endast en HTML-sida laddas initialt, varefter all vidare interaktion och dataladdning sker via asynkrona JavaScript-anrop. Vi använde Vite som byggverktyg, vilket möjliggjorde snabb utvecklingsfeedback och effektiv bundling.

Routing implementerades med React Router, där vyer som kursöversikt, kursdetaljer och inloggningssida definierades som olika rutter. Gränssnittet byggdes komponentbaserat med återanvändbara element som `CourseCard`, `CourseCardGroup` och `RoadmapCard`. Dessa komponenter används för att visa kursinformation i responsiva gridlayouter.

För design och layout användes React Bootstrap, som gav tillgång till färdiga UI-komponenter såsom navigationsfält, kort och knappar. Genom att kombinera dessa med egen CSS kunde vi snabbt skapa ett enhetligt och responsivt utseende. Ikoner från React Icons och Bootstrap Icons användes för att inkludera exempelvis sociala medie-symboler i sidfoten.

4.4 Medieintegration och materialvisning

En viktig del av plattformen är möjligheten att visa olika typer av kursmaterial direkt i webbläsaren. För videoinnehåll integrerades YouTube-videor med hjälp av `<iframe>`-element, där videons ID hämtades från databasen och inkluderades i URL:en enligt standardformatet `https://www.youtube.com/embed/VIDEO_ID`. För dokumentmaterial som PDF-filer och PowerPoint-presentationer användes HTML-elementen `<embed>` och `<object>` för att visa innehållet direkt på sidan. PowerPoint-filer konverterades till PDF vid uppladdning för att garantera kompatibilitet med alla webbläsare. Denna implementation förbättrade användarupplevelsen genom att undvika behovet av nedladdning av filer.

4.5 Autentisering och säkerhetslösning

Autentisering implementerades både som traditionell användarregistrering med e-post och lösenord, samt via Google-inloggning genom OAuth 2.0. För traditionell inloggning används hashning och saltning med algoritmer som Bcrypt eller Argon2. Inga lösenord lagras i klartext; istället beräknas ett hashvärde av lösenordet tillsammans med en unik salt-sträng och lagras i databasen.

Google-inloggning hanterades genom Googles OAuth 2.0-flöde. När en användare loggar in med sitt Google-konto returnerar Google en ID-token, som verifieras av backend. Om tokenen är giltig

loggas användaren in eller registreras baserat på profildatan. Detta möjliggör Single Sign-On och minskar behovet av att skapa ytterligare lösenord.

För sessionshantering används JSON Web Tokens (JWT). Efter lyckad inloggning returneras en signerad JWT till klienten och lagras i `localStorage`. Vid efterföljande API-anrop inkluderas tokenen i HTTP-headern, vilket gör det möjligt för backend att validera användaren utan att behöva hålla server-side-sessioner. Detta är särskilt effektivt i SPA-arkitektur.

4.6 Testning och kvalitetssäkring

För att säkerställa applikationens funktionalitet implementerades både enhetstester och integrationstester. På backend-sidan användes testbiblioteket xUnit tillsammans med Moq för att skapa mockade versioner av tjänster och repository-klasser. Detta möjliggjorde isolerad testning av logik utan beroenden till verklig databas eller API. Exempelvis kunde `CourseService` testas genom att injicera ett mockat repository som returnerade fördefinierade testdata. Detta tillvägagångssätt användes även för kontroller såsom `AuthController` och `UserController`. För integrationstester användes ASP.NET Core:s stöd för in-memory-testserver där HTTP-anrop kunde simuleras mot API:t utan att påverka produktionsdata. Testerna kördes automatiskt vid varje commit genom GitLab CI/CD, vilket hjälpte till att tidigt upptäcka regressionsfel och garantera systemets stabilitet under utveckling.

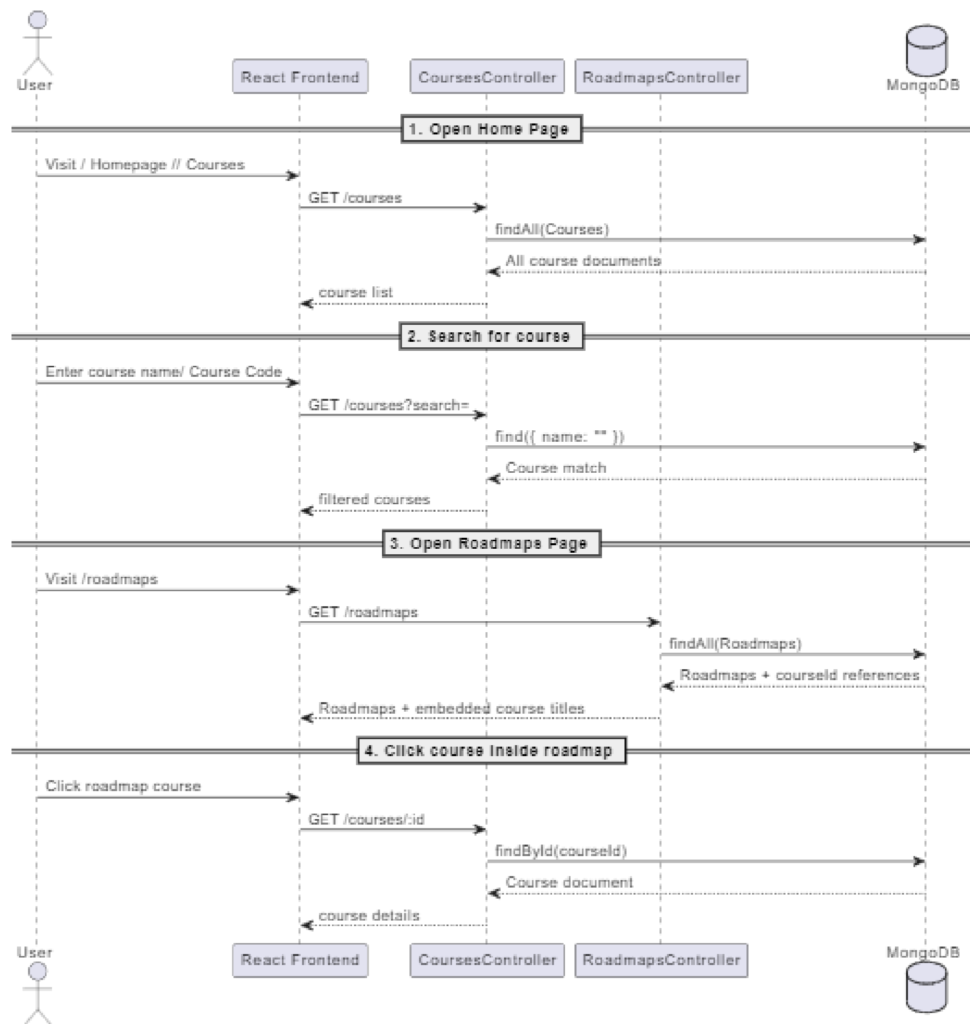
4.7 Klassdiagram

Klassdiagrammet (se Figur 6) ger en visuell översikt över backendens centrala komponenter och deras relationer. Modellerna såsom `UserModel` och `CourseModel` innehåller inbäddade listor för kapitel och progress, medan tjänstelagret använder beroendeinjektion för att få tillgång till databasen. Controllers använder tjänsterna för att exekvera affärslogik och returnerar resultatet som JSON till frontend.

4.8 Sekvensdiagram

Sekvensdiagrammen visar hur backend hanterar specifika scenarier där klienten gör anrop till API:et. Dessa flöden illustrerar samverkan mellan användare, controller, tjänst och databas.

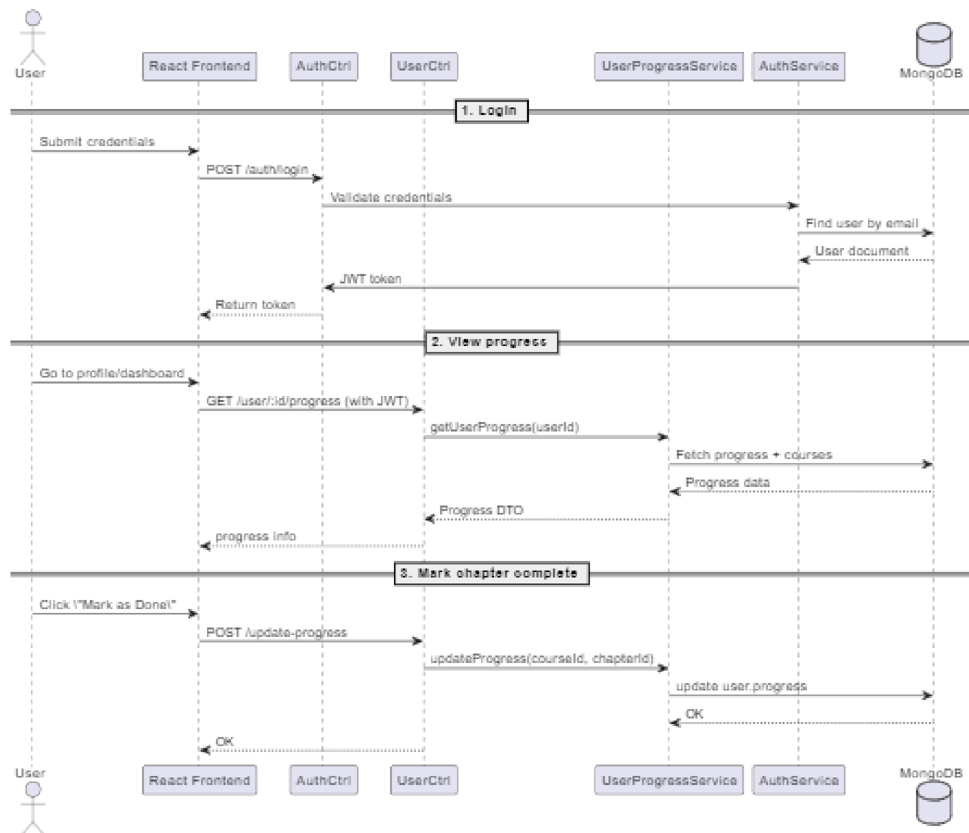
Scenario 1: Hämta en kurs via sökord



Figur 4: Sekvensdiagram: sökning efter kurs

1. Användaren skickar en `GET`–förfrågan till `/courses/search/{query}`.
2. `CoursesController` tar emot anropet och vidarebefordrar det till `CourseService`.
3. `CourseService` söker i databasen efter matchande kursnamn eller kurskod.
4. Resultatet returneras via controllern tillbaka till användaren i JSON-format.

Scenario 2: Uppdatera användarprogression



Figur 5: Sekvensdiagram: uppdatering av progression

1. Frontend skickar en `POST`–förfrågan till `/user/update-progress` med kurs- och kapitelinformation.
2. `UserController` skickar datan till `UserProgressService`.
3. Progressen uppdateras i användarens dokument i MongoDB.
4. Bekräftelse returneras som ett `200 OK`-svar.

4.9 Sammanfattning

Systemets backend är uppdelad i tydliga lager med separerade ansvarsområden, vilket underlättar vidareutveckling och testning. Arkitekturen är anpassad för REST-kommunikation och möjliggör integration med frontend via väldefinierade endpoints. Genom användningen av MongoDB som databas och ASP.NET Core för API-logik har ett skalbart och underhållbart system byggts upp. Sekvensdiagrammen visar på en tydlig kommunikationskedja mellan klient och server, där varje lager fyller en specifik funktion. Denna struktur utgör grunden för plattformens fortsatta utveckling.

5 Testning och Kvalitetssäkring

För att säkerställa systemets funktionalitet och stabilitet användes både automatiserad och manuell testning, samt kontinuerlig integration (CI) via GitLab CI/CD.

5.1 Enhetstestning

För att säkerställa affärslogikens korrekthet och minska risken för regressionsfel har ett antal enhetstester implementerats för backend-API:et. Dessa tester är isolerade från datalager och beroenden, och verifierar att varje enskild komponent beter sig korrekt under olika scenarier. Testerna är skrivna med hjälp av **xUnit**, ett välanvänt ramverk för .NET-testning, i kombination med **Moq** för att mocka externa tjänster. Denna strategi möjliggör snabb exekvering och tydliga felmeddelanden, vilket underlättar felsökning vid misslyckade byggen.

Testarkitektur Testerna är strukturerade enligt modellen *Arrange–Act–Assert*, vilket innebär:

- **Arrange:** Förbered mockade objekt, indata och förväntade resultat.
- **Act:** Anropa den metod som ska testas.
- **Assert:** Verifiera att resultatet är som förväntat.

Översikt över testklasser Följande testklasser har skapats för olika delar av systemets tjänstelager:

- `CoursesControllerTests` – Tester för hantering av kurser, inklusive sökning, hämtning och felhantering.
- `RoadmapsControllerTests` – Verifierar att roadmap-funktionalitet fungerar som förväntat.
- `AuthControllerTests` – Säkerställer att inloggningslogik, token-generering och felhantering vid autentisering är korrekt implementerad.
- `UserServiceTests` – Fokuserar på affärslogik kring användarhantering, t.ex. rollverifiering, skapande och uppdatering.
- `ProgressTrackerTests` – Testar logik för kursprogression, inklusive sparande av användarens framsteg och tillgång till nästa steg i en roadmap.

Exempel på testfall Nedan följer ett urval representativa testfall som ingår i testsviten:

- `SearchCourses_ReturnsResults_OnPartialMatch` – Testar att API:et returnerar matchande kurser även vid delvis inmatad kursnamn.
- `GetRoadmaps_ReturnsEmptyList_WhenNoneExist` – Verifierar att tomma resultat hanteras korrekt i användargränssnittet.
- `Login_ReturnsToken_OnValidCredentials` – Bekräftar att en JWT-token genereras när giltiga inloggningsuppgifter används.
- `TrackProgress_ThrowsError_IfUserUnauthorized` – Säkerställer att endast inloggade användare kan spara kursprogression.
- `CreateUser_FailsOnDuplicateEmail` – Testar att användarskapande misslyckas om e-postadressen redan finns i systemet.

Dessa testfall täcker både lyckade scenarier och felhantering, vilket ger ett robust skydd mot oväntade buggar och regressionsfel i kodbasen. Målet har varit att uppnå hög testbarhet genom tydligt ansvarsfördelade komponenter med väldefinierade gränssnitt. Alla tester körs automatiskt i samband med varje commit via projektets CI/CD-pipeline. Misslyckade tester bryter bygget och förhindrar att ny kod mergas till huvudgrenen, vilket upprätthåller kodens kvalitet över tid.

5.2 CI/CD och testautomatisering

Projektet är integrerat med **GitLab CI** för att köra enhetstester automatiskt vid varje push till huvudgrenen. CI-pipelinen är konfigurerad att:

- Installera nödvändiga beroenden.
- Bygga backend-projektet.
- Köra samtliga enhetstester.
- Logga resultatet i **XUnit-format** för enkel visualisering och uppföljning.

Exempel från `.gitlab-ci.yml`:

```
test:
  stage: test
  image: mcr.microsoft.com/dotnet/sdk:6.0
  script:
    - dotnet restore
    - dotnet build
    - dotnet test --logger:"Xunit"
```

Detta tillvägagångssätt säkerställer att kod som inte klarar testerna aldrig slås ihop i produktionsgrenen, vilket förbättrar systemets övergripande kvalitet och minskar risken för regressionsfel.

5.3 Manuell testning av frontend

Den React-baserade frontend-delen testades manuellt genom direkt interaktion i webbläsaren. Fokus låg på att säkerställa:

- Funktionella flöden (t.ex. inloggning, navigering, kurshantering).
- Felhantering och tydlig användaråterkoppling.
- Responsivitet i mobilläge.

Tester genomfördes iterativt under utveckling och vid varje större feature-release.

6 Resultat

Det färdiga systemet för *TestScouts University* består av två huvudsakliga komponenter: ett backend-API utvecklat i ASP.NET Core och en frontend byggd med React. Tillsammans utgör dessa en fungerande helhet för en utbildningsplattform där användare kan logga in, navigera mellan kurser, följa roadmap-strukturer och se sin individuella progression. Systemet har genomgått både manuell och automatisk testning och uppvisar stabil funktionalitet i alla centrala användarflöden.
Backend-API:et möjliggör bland annat:

- Hämtning och sökning av kurser, med stöd för sökfrågor på namn eller kod.
- Hämtning av roadmap-strukturer som beskriver inbördes relationer mellan kurser.
- Registrering och autentisering via e-post eller Google OAuth, med säker token-hantering genom JWT.
- Uppdatering av användarens kapitelprogression, där varje genomfört avsnitt loggas till MongoDB.

Systemets REST-endpoints har utvecklats med tydliga och konsistenta namngivningar, asynkrona metoder och korrekt statuskodshantering. API:et är uppdelat i controllers, tjänster (services) och modeller, vilket skapar en modulär och testbar arkitektur. MongoDB används som databas, där användare, kurser och roadmap-strukturer lagras i separata collections. Datamodellerna är försedda med `BsonElement`-attribut för att säkerställa struktur trots den schemafria naturen i MongoDB.

Frontend-komponenten är en React-applikation som använder Axios för att kommunicera med backend. Frontendens roll är att presentera data från API:et på ett användarvänligt och responsivt sätt. Användare kan logga in, se tillgängliga kurser, följa roadmap-strukturer och få en visuell översikt av sin progression. React-komponenter är återanvändbara och uppbyggda enligt modern designpraxis, vilket gör gränssnittet både skalbart och underhållsbart.

Systemet har testats genom:

- **Manuell testning:** Samtliga endpoints verifierades i Postman med varierande indata, inklusive felaktiga och ogiltiga parametrar.
- **Automatiserad testning:** Ett antal enhetstester och integrationstester körs automatiskt via GitLab CI vid varje push till huvudgrenen.
- **Frontend-testning:** Testanrop från React har verifierats genom faktiska användarscenarier som inloggning, navigering och progression.

Resultaten visar att systemet är stabilt, korrekt integrerat och uppfyller kraven för ett MVP (Minimum Viable Product). Funktionaliteten motsvarar projektets mål, och det tekniska fundamentet är tillräckligt robust för vidareutveckling – inklusive framtida utbyggnad av adminfunktioner, rollstyrning och avancerad statistik. —

7 Diskussion

Projektet har resulterat i ett fungerande backend-API som uppfyller många av de funktionella krav som definierades i början. Den övergripande systemdesignen, med tydlig separation av ansvar mellan controllers, tjänster och modeller, har visat sig vara en hållbar och skalbar lösning. Det REST-baserade API:et ger god flexibilitet vid integration med olika typer av frontendklienter, vilket är viktigt för ett växande system som potentiellt kan utökas med fler användarroller eller funktioner.

Trots de positiva resultaten finns flera aspekter att reflektera över. Under utvecklingsfasen identifierades vissa tekniska begränsningar, särskilt kopplat till MongoDB:s schemafria natur. Även om denna flexibilitet varit en fördel under tidig utveckling, medför det också en ökad risk för inkonsekvent datalagring. Lösningen har varit att använda tydliga datamodeller och validering i backend, men framtida utbyggnader kan kräva striktare schemahantering.

En annan viktig aspekt att diskutera är testbarhet. Även om vissa automatiserade tester integrerats i CI/CD-processen, saknas idag fullständig testtäckning för flera kritiska delar av applikationen. Exempelvis finns begränsad täckning för felhantering vid databaskrascher eller JWT-validering under speciella förhållanden. Detta utgör en potentiell felkälla i produktionsmiljöer och bör åtgärdas genom mer omfattande enhets- och integrationstester.

Det bör också nämnas att viss funktionalitet, såsom skapande och redigering av kurser och roadmap-strukturer, i dagsläget saknar en administrativ gränssnitt och kräver direkt åtkomst till databasen eller API:et. Detta utgör en begränsning för systemets användbarhet i praktiken

och är en viktig punkt för framtida vidareutveckling.

Sammanfattningsvis har projektet uppnått ett stabilt minimum av funktionalitet, men för att kunna användas i en verklig lärplattform krävs ytterligare steg, särskilt inom testning, behörighetskontroll och administrativ hantering.

8 Framtida arbete

Flera förbättringsområden har identifierats under arbetets gång som bör prioriteras i framtida utveckling:

- **Administrativa funktioner:** Systemet saknar idag ett admin-gränssnitt för att hantera kurser, roadmap-strukturer och användare. Att implementera rollbaserad åtkomstkontroll samt skapa specifika endpoints och vyer för administratörer är en naturlig nästa utvecklingsfas.
- **Rollhantering och åtkomstkontroll:** En mer avancerad behörighetsmodell, med stöd för roller som student, lärare och admin, bör införas. Detta möjliggör granular kontroll över vilka endpoints och funktioner varje användare har tillgång till.
- **Testtäckning och felhantering:** För att uppnå produktionsstandard krävs utökad testning, både i form av enhetstester och integrationstester. Felhantering bör också förbättras, särskilt vid oväntade systemfel eller ogiltiga användarinmatningar.
- **Statistik och rapporter:** Funktioner för att samla in och visualisera statistik kring användarprogression, kursavslut och engagemang skulle förbättra plattformens värde för lärare och administratörer.

Dessa förbättringar kräver både tekniska justeringar i backend samt design av nya gränssnitt i frontend. Med en solid grund etablerad i detta projekt är systemet väl förberett för fortsatt utbyggnad.

Referenser

[1] Microsoft Learn. "Best practices for ASP.NET Core applications." Accessed: 2025-07-28. (2024), URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/best-practices?view=aspnetcore-9.0.

[2] L. Veiga. "Building RESTful APIs with .NET8: Essential Best Practices for Performance and Security." Accessed: 2025-07-28. (2024), URL: https://dev.to/leandroveiga/building-restful-apis-with-net-8-essential-best-practices-for-performance-and-security-519l.

[3] MongoDB Developer. "Create a RESTful API With .NET Core and MongoDB." Accessed: 2025-07-28. (2024), URL: https://www.mongodb.com/developer/languages/csharp/create-restful-api-dotnet-core-mongodb/.

[4] J. Watmore. "ASP.NET Core 3 - Hash and Verify Passwords with BCrypt." Accessed: 2025-07-28. (2020), URL: https://jasonwatmore.com/post/2020/07/16/aspnet-core-3-hash-and-verify-passwords-with-bcrypt.

[5] GitLab. "Benefits of Continuous Integration." Accessed: 2025-05-19. (2024), URL: https://about.gitlab.com/topics/ci-cd/benefits-continuous-integration/.

[6] Red Hat. "What is CI/CD?" Accessed: 2025-05-19. (2024), URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd.

[7] GitLab. "CI/CD pipelines | GitLab Docs." Accessed: 2025-05-19. (2024), URL: https://docs.gitlab.com/ci/pipelines/.

[8] GitLab. "Unit test report examples | GitLab Docs." Accessed: 2025-05-19. (2024), URL: https://docs.gitlab.com/ci/testing/unit_test_report_examples/.

[9] GitLab, *Canary deployments*, Accessed: 2025-05-19, 2024. URL: https://docs.gitlab.com/user/project/canary_deployments/.

[10] Codefresh, *What is Canary Deployment?* Accessed: 2025-05-19, 2024. URL: https://codefresh.io/learn/software-deployment/what-are-canary-deployments/.

[11] Anonymous, "React: A Detailed Survey," *International Journal of Computer Applications*, 2023, Accessed: 2025-07-30. DOI: 10.XXXX/ReactSurvey2023. URL: https://www.researchgate.net/publication/361085649_React_A_detailed_survey.

[12] M. A. S. Dr. Vishal Shrivastava Dr. Akhil Pandey, "React JS – A Frontend JavaScript Library," *International Journal of Recent Publication Research*, 2022, Accessed: 2025-07-30. DOI: 10.XXXX/IJRPRReact2022. URL: https://ijrpr.com/uploads/V5ISSUE3/IJRPR23525.pdf.

[13] H.-H. U. of Applied Sciences, "A Comparative Analysis of Webpack and Vite as Build Tools," *Master's Thesis, Theseus.fi*, 2024, Accessed: 2025-07-30. URL: https://www.theseus.fi/bitstream/handle/10024/877513/Nguyen_TuanAnh.pdf.

[14] Contentful Blog. "Mastering React routing: A guide to routing in React." Accessed: 2025-07-25. (2025), URL: https://www.contentful.com/blog/react-routing/.

[15] Great Frontend Blog. "Most Useful and Impactful React Ecosystem Libraries." Accessed: 2025-07-15. (2024), URL: https://www.greatfrontend.com/blog/most-useful-and-impactful-react-ecosystem-libraries.

[16] React Icons GitHub. "React Icons – Include popular icons in your React projects easily with ES6 imports." Accessed: 2025-07-20. (2025), URL: https://react-icons.github.io/react-icons/.

[17] npm Package Registry. “react-circular-progressbar – A circular progressbar component, built with SVG and extensively customizable.” Accessed: 2025-07-28. (2025), URL: https://www.npmjs.com/package/react-circular-progressbar.

[18] M. Szewczyk och M. Skublewska-Paszkowska. “Performance Comparison of Java EE and ASPET Core Technologies for Web API Development.” Accessed: 2025-07-20. (2025), URL: https://www.researchgate.net/publication/325534947_Performance_Comparison_of_Java_EE_and_ASPNET_Core_Technologies_for_Web_API_Development.

[19] Vite Contributors. “Vite – Next Generation Frontend Tooling.” Accessed: 2025-07-22. (2025), URL: https://vite.dev/.

[20] A. Hodijah och U. T. Setijohatmo. “Experimental Evaluation of Relationships Model Between Documents in MongoDB.” Accessed: 2025-07-21. (2020), URL: https://www.atlantis-press.com/article/125949764.pdf.

[21] D. Fett, R. Küsters och G. Schmitz. “A Comprehensive Formal Security Analysis of OAuth 2.0.” Accessed: 2025-07-22. (2016), URL: https://arxiv.org/abs/1601.01229.

[22] O. Etese och A.-A. Adesina. “A Review and Comparative Analysis of Password Hashing Techniques: Evaluating Bcrypt and Argon2.” Accessed: 2025-07-22. (2025), URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5363433.

[23] Microsoft Docs. “Best practices for writing unit tests - .NET.” Accessed: 2025-07-22. (2025), URL: https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices.

[24] Microsoft Docs. “Integration tests in ASPET Core.” Accessed: 2025-07-22. (2025), URL: https://learn.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-9.0.

[25] D. Esther. “Automated Testing Strategies for Quality Assurance in CI/CD Pipelines.” Accessed: 2025-07-18. (2024), URL: https://www.researchgate.net/publication/385286073_Automated_Testing_Strategies_for_Quality_Assurance_in_CICD_Pipelines.

Bilagor

Bilaga A: Översikt över REST-endpoints

Endpoint	Beskrivning
`GET /api/courses`	Hämtar en lista med alla tillgängliga kurser
`GET /api/courses/search?query=`	Söker efter kurser baserat på namn eller kurskod
`GET /api/courses/{id}`	Hämtar detaljerad information om en specifik kurs
`GET /api/roadmaps`	Hämtar alla tillgängliga roadmap-strukturer
`POST /api/user/update-progress`	Uppdaterar användarens progression (markerar kapitel som slutfört)
`GET /api/user/{id}/progress`	Hämtar en användares kursprogression och kapitelstatus
`POST /auth/login`	Autentiserar användare med e-post och lösenord (lokal inloggning)
`POST /auth/google`	Autentiserar användare via Google OAuth och returnerar JWT-token
`POST /auth/register`	Registrerar en ny användare (lokal registrering)

Bilaga B: Översikt över klasser

Figur 6: Klassdiagram för centrala backendkomponenter

DEPARTMENT OF COMPUTING SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS